Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Samo Pahor

NP-polnost miselnih iger in ugank

diplomsko delo
univerzitetni študijski program prve stopne
računalništvo in matematika

Mentor: doc. dr. Jurij Mihelič

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Neobvladljivost oz. NP-polnost odločitvenih problemov je eden izmed najpomembnejših konceptov v teoretičnem računalništvu. Znano je, da so NP-polni problemi velik izziv za učinkovito reševanje. V okviru diplomske naloge opišite teoretično ozadje neobvladljivosti in opravite pregled miselnih iger in ugank, ki so že bile dokazane kot NP-polne. Izbrane uganke opišite in formalno predstavite pripadajoče odločitvene probleme. Podrobno predstavite dokaz NP-polnosti uganke in vse pripadajoče podrobnosti prevedbe.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Samo Pahor sem avtor diplomskega dela z naslovom:

NP-polnost miselnih iger in ugank (angl. *NP-completeness of puzzles*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Jurija Miheliča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 28. januarja 2016 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Juriju Miheliču za pomoč in usmerjanje ter kopico uporabnih komentarjev.

Prav tako se zahvaljujem prof. Mariji Šubic, ki je delo lektorirala.

Za konec se zahvaljujem še družini in prijateljem, ki so mi stali ob strani.

Kazalo

Povzetek

Najtežje probleme v NP imenujemo NP-polni problemi, za njih pa trenutno velja, da jih ne znamo rešiti v polinomskem času. NP-polne odločitvene probleme najdemo na različnih področjih, od teorije grafov, do izjavne logike, nadalje pa velja, da ob odkritju rešitve enega od problemov znamo rešiti tudi vse druge NP-polne probleme. V diplomskem delu se bomo osredotočili predvsem na NP-polne probleme, povezane z reševanjem miselnih iger in ugank. Naredili bomo pregled nekaterih znanih miselnih iger in pokazali njihovo NP-polnost. Dokaz NP-polnosti izbranih problemov bo temeljil na dejstvu, da med NP-polnimi problemi obstajajo prevedbe. Za vsako miselno igro bomo torej najprej opisali igri pripadajoči odločitveni problem, nato pa reševanje nekega že znanega NP-polnega odločitvenega problema prevedli na reševanje igri pripadajočega odločitvenega problema.

Ključne besede: odločitveni problem, uganka, računska zahtevnost, NP-polnost, prevedba.

Abstract

The hardest problems in NP are called NP-complete problems; these are problems that we currently do not know how to solve quickly. There is a variety of NP-complete decision problems in different fields, ranging from graph theory to logic and also note that upon finding a solution to one NP-complete problem, we are able to solve any problem that belongs to the NP-complete class of problems. The thesis will focus on NP-complete problems related to solving puzzles and brain-teasers. We will present a summary of a few well known puzzles and show that they are indeed NP-complete. The proof for their NP-completeness will stem from the existence of transformations (or reductions) between them. For each puzzle we will first present the respective decision problem and then transform the solving of an already proven NP-complete decision problem to solving the decision problem pertaining to each puzzle.

Keywords: decision problem, puzzle, computational complexity, NP-completeness, reduction.

Poglavje 1

Uvod

Ljudje že od nekdaj radi rešujemo uganke. Privlačijo nas miselni izzivi in zanimivo zastavljeni problemi, pogosto pa se ugank lotimo preprosto zato, da bi pregnali dolgčas. Same uganke obstajajo v različnih oblikah in zahtevnostih; od preprostih vprašanj z domiselnimi odgovori do zapletenih miselnih zagonetk s kompliciranimi pravili in sistemom reševanja. Predvsem slednje predstavljajo izziv ne samo rekreativnim reševalcem, temveč tudi raziskovalcem, ki v želji po večjem razumevanju problemov, povezanih z ugankami, razvijajo algoritme in metode za njihovo reševanje. V nasprotju z rekreativci se raziskovalci običajno ne trudijo rešiti zgolj ene same variante uganke, temveč skušajo razviti algoritem, ki bi dokazal rešljivost oziroma nerešljivost celotnega razreda ugank.

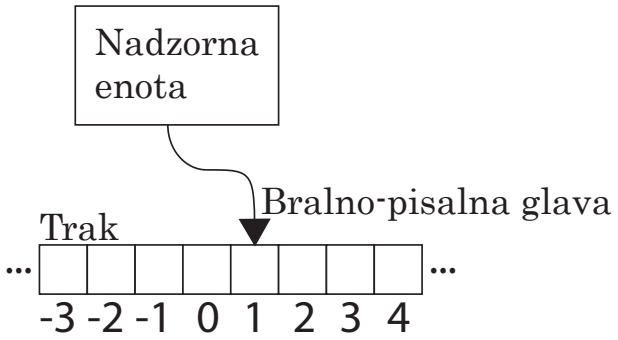
Diplomska naloga predstavlja pregled nekaterih NP-polnih miselnih iger in ugank. V drugem poglavju najprej naredimo pregled teoretičnega ozadja potrebnega za razumevanje koncepta NP-polnosti, nato pa sledi poglavje, kjer pregledamo izbrani seznam NP-polnih ugank. Pri vsaki uganki opišemo njeno ozadje in izvor, definiramo njena pravila in dokažemo NP-polnost uganke. Po pregledu vseh ugank sledi zaključek, kjer komentiramo pregledano področje in predlagamo podobna področja primerna za raziskovanje.

Poglavje 2

Teoretično ozadje

Za razumevanje jedra diplomske naloge si bomo pred tem ogledali potrebno teoretično ozadje. Na kratko bomo opisali deterministični in nedeterministični Turingov stroj, razreda problemov P in NP ter odnos med njima. Razložili bomo pomen prevedb in pojem odločitvenega problema ter na kratko opisali prvi znani NP-poln problem — SAT.

2.1 Deterministični Turingov stroj



Slika 2.1: Deterministični Turingov stroj

V računalništvu probleme rešujemo z algoritmi. Slednji so procesi oziroma seznami pravil, ki v končnem številu korakov pripeljejo do rešitve problema. Za formalno predstavitev samega koncepta algoritma je potrebno prej definirati model računanja. Med standardne univerzalne modele računanja uvrščamo Von Neumannov model, model Markova, Postov model in druge, v tem delu pa se bomo predvsem osredotočili na **deterministični Turingov stroj** (na kratko DTS), (angl. *deterministic Turing machine*). DTS sestavljajo trije ključni elementi (slika 2.1):

- nadzorna enota, ki vsebuje končno mnogo stanj in je v vsakem trenutku v natanko enem od teh stanj,
- dvosmerno neomejen trak, sestavljen iz neskončnega števila celic,
- bralno-pisalna glava, ki vidi natanko eno celico na traku in se lahko premika levo oziroma desno po eno celico naenkrat. Glava lahko iz celice, na kateri se nahaja, prebere simbol in v celico zapiše drug ali isti simbol.

Program DTS lahko formalno definiramo kot sedmerko $\langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, za katero velja:

1. Q predstavlja končno, neprazno množico stanj.
2. Γ predstavlja končno, neprazno množico tračnih simbolov.
3. $b \in \Gamma$ predstavlja prazen simbol.
4. $\Sigma \subseteq \Gamma \setminus \{b\}$ predstavlja končno, neprazno množico vhodnih simbolov.
5. $\delta\colon (Q - \{q_Y, q_N\}) \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ je funkcija prehodov.
6. $q_0 \in Q$ je začetno stanje.
7. $F \subseteq Q$ je množica končnih oziroma zadovoljivih stanj.

Delovanje programa na DTS je preprosto. DTS na vhodu sprejme vhodno besedo $x \in \Sigma^* (= \bigcup_{n=0}^{\infty} \Sigma^n)$, sestavljeno iz vhodnih simbolov. Ti simboli se po vrsti zapišejo v celice od 1 do $|x|$. Vse ostale celice na traku imajo v celicah zapisan prazen simbol b.

Program začne svoje delovanje v začetnem stanju q_0, bralno-pisalna glava pa začne branje pri prvi celici. Delovanje nato poteka po korakih in se zaključi, ko stanje q postane bodisi q_Y (odgovor na vhodno besedo je "da") bodisi q_N (odgovor na vhodno besedo je "ne"). Spodaj je opisan primer enega koraka delovanja programa. Koraki seveda potekajo eden za drugim in so odvisni od funkcije prehajanja stanj δ. Denimo, da se nahajamo v stanju q, bralno-pisalna glava pa se nahaja na celici s simbolom s. Nadalje velja:

$$\delta(q, s) = (q', s', \Delta)$$

Zgornji zapis pomeni, da se notranje stanje DTS spremeni iz q v q', bralno-pisalna glava pa simbol s na trenutno opazovani celici pobriše in zamenja s s'. Bralno-pisalna glava se nato premakne v smeri $\Delta \in \{-1, +1\}$. To pomeni, da se premakne za eno celico v levo, če je $\Delta = -1$, oziroma eno celico v desno, če je $\Delta = +1$.

V splošnem rečemo, da program M Turingovega stroja z množico vhodnih simbolov Σ *sprejme* vhodno besedo $x \in \Sigma^*$, če in samo če se M zaustavi v stanju q_Y in je bila njegova vhodna beseda x. Formalno lahko definiramo jezik Turingovega stroja L_M, ki ga *prepozna* program M in zanj velja:

$$L_M = \{x \in \Sigma^* : \text{M } \textit{sprejme} \text{ x}\}$$

Opazimo, da ni potrebno, da se M zaustavi pri vseh vhodnih besedah v Σ^*, temveč zgolj pri tistih, ki so v L_M. Če x pripada $\Sigma^* - L_M$, potem se izračun lahko zaustavi v stanju q_N ali pa se nikoli ne zaustavi. Za DTS program rečemo, da je *algoritem* natanko takrat, ko se zaustavi za vse možne vhodne besede iz Σ^*.

2.2 Odločitveni problem

Za razumevanje vsebine dela moramo definirati koncept (odločitvenega) problema, saj bomo v prihajajočih razdelkih pregledovali predvsem različne odločitvene probleme in povezave med njimi.

Odločitveni problem je vprašanje, ki na podlagi vhodnih informacij lahko vrne bodisi odgovor "da" bodisi "ne". Sestavljen je iz dveh delov:

- splošnega opisa parametrov,
- izjave, ki opisuje, katere lastnosti mora izpolniti rešitev problema.

Primer odločitvenega problema je npr. vprašanje: "Ali je podano število x praštevilo (torej število večje od 1, ki je deljivo zgolj z 1 in samo s sabo)?"

Rešljivost problema enačimo z obstojem algoritma, ki izbrani problem reši. Pri zgornjem odločitvenem problemu je eden od možnih algoritmov, ki ga lahko uporabimo za reševanje, algoritem zaporednih deljenj. Pri slednjem algoritmu po vrsti preverjamo ali je podano število n deljivo s kakim od števil

med 2 in $\sqrt{n}$. Če n ni deljivo z nobenim od omenjenih števil, potem gre za praštevilo, sicer pa je n sestavljeno število.

Dani odločitveni problem je *odločljiv* oziroma *izračunljiv*, saj za vsako število vrne bodisi "da" bodisi "ne". Ta označba sprva morda izgleda trivialno, vendar poznamo veliko odločitvenih problemov, ki niso izračunljivi.

Za te probleme velja, da so *neizračunljivi* oziroma *neodločljivi*, saj za njih ne obstaja algoritem, ki lahko reši problem. Primer takega problema je problem ustavljivosti, predstavljen v delu Alana Turinga "On computable numbers, with an application to the Entscheidungsproblem" [1], ki se glasi:

Problem 1. *Ali se podani Turingov stroj T pri podanih vhodnih podatkih i zaustavi?*

Na poljubno izbrani način zakodirani odločitveni problem torej predstavlja problem pripadnosti dane besede nekemu formalnemu jeziku. Lahko bi rekli, da je odločitveni problem visokonivojska predstavitev jezika Turingovega stroja.

2.3 Razred P

Za DTS, ki smo ga definirali v prvem razdelku poglavja, lahko definiramo pojem *časovne zahtevnosti*. Čas, ki ga porabimo pri izračunu DTS programa M pri vhodni besedi x, je enak številu korakov, ki nas privedejo do zaustavitve. Za DTS program M, ki se zaustavi za vse vhodne besede $x \in \Sigma^*$, je *funkcija časovne zahtevnosti* $T_M \colon Z^+ \to Z^+$ definirana takole:

$$T_M(n) = \max \left\{ m \colon \begin{array}{l} \text{obstaja } x \in \Sigma^* \text{ in } |x| = n, \text{ tako da izračun} \\ M \text{ pri vhodni besedi } x \text{ vzame } m \text{ korakov} \end{array} \right\}$$

Za program M rečemo, da ima *polinomsko časovno zahtevnost*, če obstaja tak polinom p, da velja $T_M(n) \leq p(n)$ za vsak $n \in Z^+$.

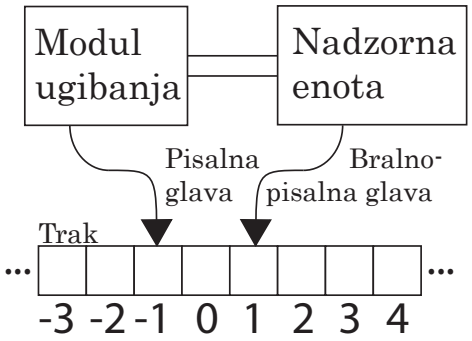
Sedaj lahko formalno definiramo prvi razred jezikov, ki ga bomo opazovali. Imenuje se razred P, definiran pa je takole:

$$P = \left\{ L: \begin{array}{l} \text{obstaja DTS program } M \text{ polinomske časovne} \\ \text{zahtevnosti, za katerega velja } L = L_M \end{array} \right\}$$

2.4 Nedeterministični Turingov stroj

Na podoben način, kot smo v prejšnjem razdelku definirali razred P, bomo v naslednjem razdelku definirali razred NP. Preden pa se lahko lotimo same definicije, pa moramo definirati **nedeterministični Turingov stroj** (NDTS) (angl. *nondeterministic Turing machine*).

NDTS ima povsem enako strukturo kot DTS, dodamo mu zgolj *modul ugibanja* s svojo lastno pisalno glavo (slika 2.2). Tudi NDTS program deluje na isti način kot DTS, razlikujeta se zgolj v tem, da pri NDTS izračun poteka v dveh fazah.



Slika 2.2: Nedeterminstični Turingov stroj

Prva je faza "ugibanja", kjer se na trak od celice 1 do $|x|$ zapišejo simboli vhodne besede x, bralno-pisalna glava je pri celici 1, pisalna glava modula ugibanja pa pri celici -1. Nadzorna enota ni aktivna. Modul ugibanja nato usmerja pisalno glavo, ki v trenutno celico zapiše nek simbol iz Γ in se premakne za eno celico v levo, ali pa se ustavi in z delovanjem zaključi. Modul

ugibanja lahko na trak napiše poljuben niz iz $\Gamma^* = \bigcup_{n=0}^{\infty} \Gamma^n$, kar pomeni, da se lahko nikoli ne ustavi.

Če se modul ugibanja zaustavi, se aktivira nadzorna enota in začne fazo "preverjanja". V tej fazi modul ugibanja in pisalna glava nista aktivna in NDTS program deluje po natanko istih pravilih kot DTS. Izračun se zaključi, ko in če stanje postane bodisi q_Y bodisi q_N. V primeru q_Y rečemo, da je prišlo do sprejemljivega, pri q_N pa do nesprejemljivega izračuna. Opazimo, da ima program NDTS za dano vhodno besedo x neskončno mnogo izračunov, enega za vsak niz iz Γ^*. Rečemo, da NDTS *sprejme* x, če je vsaj eden od izračunov sprejemljiv. Jezik L_M, ki ga *prepozna* program NDTS M, je definiran enako kot pri DTS.

2.5 Razred NP

Čas, ki ga NDTS program M potrebuje za sprejem vhodne besede $x \in L_M$, je definiran kot minimum števila korakov, ki jih v fazah "ugibanja" in "preverjanja" naredijo vsi sprejemljivi izračuni M za x. Funkcija časovne zahtevnosti $T_M \colon Z^+ \to Z^+$ je definirana takole:

$$T_M(n) = \max \left\{ \{1\} \cup \left\{ m \colon \begin{array}{l} \text{obstaja } x \in L_M \text{ in } |x| = n, \\ \text{ki ga } M \text{ sprejme v času } m \end{array} \right\} \right\}$$

Program NDTS M ima polinomsko časovno zahtevnost, če obstaja polinom p, tako da velja $T_M(n) \leq p(n)$ za vsak $n \leq 1$.

Sedaj lahko definiramo drugi razred jezikov oziroma problemov. Poimenovali ga bomo razred NP, zanj pa bo veljalo:

$$\mathrm{NP} = \left\{ L \colon \begin{array}{l} \text{obstaja NDTS program } M \text{ polinomske časovne} \\ \text{zahtevnosti, za katerega velja } L = L_M \end{array} \right\}$$

Preden lahko definiramo še zadnji razred problemov, si moramo ogledati razmerje med razredoma P in NP. Najprej opazimo, da očitno velja $\mathrm{P} \subseteq \mathrm{NP}$.

Lema 2.5.1. *Iz* $\Pi \in P$ *sledi* $\Pi \in NP$.

Dokaz. Vsak problem, ki ga lahko reši deterministični algoritem v polinomskem času, je v polinomskem času rešljiv tudi z nedeterminističnim algoritmom. Izberemo si poljubni deterministični algoritem A, ki v polinomskem času reši problem $\Pi \in$ P. Sedaj lahko skonstruiramo nedeterministični algoritem tako, da v fazi "preverjanja" preprosto poženemo algoritem A, fazo "ugibanja" pa zanemarimo. To pomeni, da iz $\Pi \in$ P sledi $\Pi \in$ NP. □

Obratne vsebovanosti NP $\subseteq$ P ne znamo ne dokazati, ne ovreči.

2.6 Ali je P=NP?

Vprašanje, ali je razred P enak razredu NP, velja za enega najpomembnejših odprtih problemov na področju teoretičnega računalništva. Bistvo problema je bilo prvič omenjeno leta 1956 v pismu Kurta Gödela Johnu von Neumannu [2], formalna definicija problema pa je bila prvič predstavljena v delu Stephena Cooka z naslovom "The complexity of theorem proving procedures" [3]. Kljub temu, da je problem star več kot 40 let, še vedno ostaja nerešen [4], matematični inštitut Clay pa ga je uvrstil med sedem t. i. *Millenium Prize Problems* in za njegovo rešitev razpisal nagrado v velikosti milijon dolarjev.

2.7 Prevedbe

Za opazovanje in določanje lastnosti različnih problemov moramo definirati pojem polinomske prevedbe (angl. *polynomial transformation*). Polinomska prevedba jezika $L_1 \subseteq \Sigma_1^*$ v jezik $L_2 \subseteq \Sigma_2^*$ je funkcija $f\colon \Sigma_1^* \to \Sigma_2^*$, ki ustreza naslednjima pogojema:

1. Obstaja program DTS s polinomsko časovno zahtevnostjo, ki izračuna f.
2. Za vsak $x \in \Sigma_1^*$ velja $x \in L_1$, če in samo če je $f(x) \in L_2$.

Če obstaja polinomska prevedba iz L_1 v L_2, potem pišemo $L_1 \propto L_2$. Polinomsko prevedbo potrebujemo zaradi spodnje leme.

Lema 2.7.1. *Če velja $L_1 \propto L_2$, potem iz $L_2 \in P$ sledi $L_1 \in P$ (in ekvivalentno iz $L_1 \notin P$ sledi $L_2 \notin P$). (Dokaz leme izpustimo, lahko pa si ga ogledamo v [5])*

Za polinomsko prevedbo prav tako velja, da je tranzitivna, o čemer govori naslednja lema.

Lema 2.7.2. *Če velja $L_1 \propto L_2$ in $L_2 \propto L_3$, potem velja $L_1 \propto L_3$. (Dokaz leme izpustimo, lahko pa si ga ogledamo v [5])*

2.8 Razred NP-polnih problemov

Formalno je jezik L NP-poln, če velja $L \in$ NP in za vse jezike $L' \in$ NP velja $L' \propto L$. Odločitveni problem Π je NP-poln, če je $\Pi \in$ NP in za vse druge odločitvene probleme $\Pi' \in$ NP velja $\Pi' \propto \Pi$. Glede na lemo 2.7.1 lahko NP-polne probleme označimo kot "najtežje probleme v NP". Naslednja lema nam pomaga za poljuben problem ugotoviti, če pripada razredu NP-polnih problemov.

Lema 2.8.1. *Če L_1 in L_2 pripadata NP, L_1 je NP-poln in velja $L_1 \propto L_2$, potem je L_2 NP-poln.*

Dokaz. Ker velja $L_2 \in$ NP, moramo pokazati, da za vsak $L' \in$ NP velja $L' \propto L_2$. Ker je L_1 NP-poln, mora veljati $L' \propto L_1$. Iz tranzitivnosti $\propto$ in dejstva, da velja $L_1 \propto L_2$, sledi $L' \propto L_2$ za vsak $L' \in$ NP. □

Z zgornjo lemo si sedaj lahko praktično pomagamo pri dokazovanju NP-polnosti izbranih problemov, ko imamo vsaj en problem, za katerega vemo, da je NP-poln. Da za poljuben problem Π dokažemo NP-polnost, moramo pokazati naslednje:

1. $\Pi \in$ NP,
2. nek znan NP-poln problem Π' lahko prevedemo na Π.

2.9 Problem SAT

Kot smo omenili zgoraj, za dokazovanje NP-polnosti poljubnega problema najprej potrebujemo problem, za katerega vemo, da je NP-poln. Naziv prvega NP-polnega problema pripada problemu izpolnjivosti (angl. *satisfiability problem*), ki mu na kratko rečemo SAT. Preden lahko problem SAT formalno podamo, moramo opisati sledeče koncepte.

Naj bo $U = \{u_1, u_2, ..., u_m\}$ množica Boolovih spremenljivk. *Resničnostna nastavitev* za U je funkcija $t\colon U \to \{T, F\}$ (T in F predstavljata vrednosti *resnično* in *neresnično*). Če velja $t(u) = T$, potem rečemo, da je u resnična pod t, v primeru $t(u) = F$ pa seveda rečemo, da je u neresnična pod t. Če je u v U, potem sta u in $\overline{u}$ *literala* nad U. Literal u je resničen pod t, če in samo če je spremenljivka u resnična pod t; literal $\overline{u}$ je resničen, če in samo če je spremenljivka u neresnična.

Stavek nad U je množica literalov nad U, na primer $\{u_1, \overline{u_2}, u_8\}$. Predstavlja disjunkcijo vsebovanih literalov in je *izpolnjen* z resničnostno nastavitvijo, če in samo če je vsaj eden od vsebovanih literalov resničen pod to resničnostno nastavitvijo. Omenjeni stavek bo izpolnjen s t, razen če velja $t(u_1) = F$, $t(\overline{u_2}) = T$ in $t(u_8) = F$. Množica stavkov C nad U je izpolnjena, če in samo če obstaja resničnostna nastavitev za U, ki naenkrat izpolni vse stavke v C. Problem SAT sedaj lahko opišemo takole:

Problem 2. *Če imamo množico spremenljivk U in množico stavkov C nad U, ali obstaja izpolnljiva resničnostna nastavitev za C?*

Dokaz NP-polnosti SAT bomo izpustili, nahaja pa se v [5].

Poglavje 3

Uganke

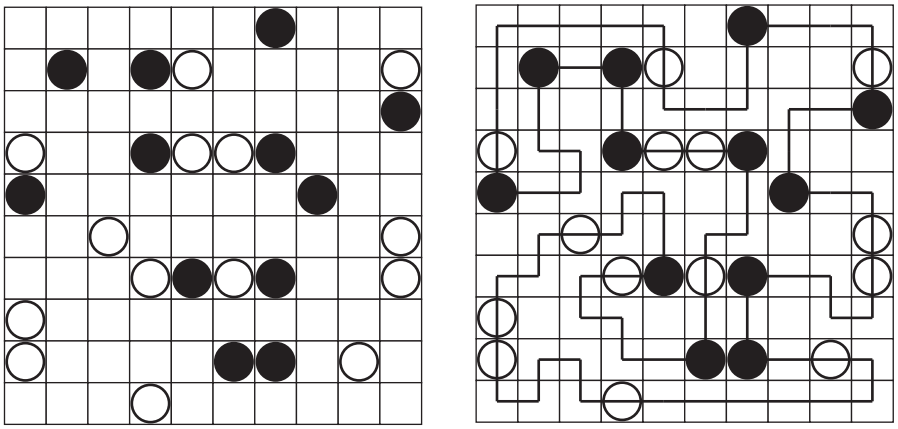
Sledi pregled NP-polnih miselnih iger in ugank. Za izbrano miselno igro ali uganko rečemo, da je NP-polna takrat, ko program NDTS M za ugotovitev *rešljivosti* porabi polinomski čas. Miselnih iger torej ne bomo reševali, temveč zgolj ugotavljali rešljivost. Vsako miselno igro bomo predstavili v treh delih. Na kratko bomo:

- predstavili ozadje in podatke o igri,
- opisali pravila igre,
- dokazali NP-polnost.

Dokaz NP-polnosti bo sestavljen iz dveh delov:

- pokazali bomo vsebovanost v NP,
- obstoječ NP-poln problem bomo prevedli na problem reševanja dane miselne igre.

3.1 Masyu



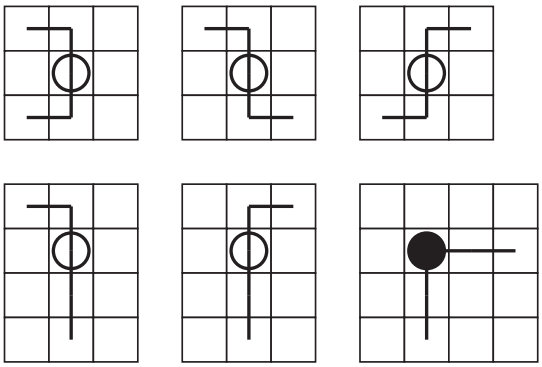
Slika 3.1: Začetno stanje Masyu uganke (levo) in podana rešitev (desno) [9]

3.1.1 Podatki o igri

Masyu (angl. *Pearls*) je japonska miselna igra, ki jo je razvilo podjetje Nikoli. Igra se je prvič pojavila v 84. številki revije *Puzzle Communication Nikoli*, takrat pa se je imenovala *Shinju no Kubikazari*, kar pomeni "biserna ogrlica". Takšno ime je imela zato, ker so v njej nastopali samo beli kamenčki. Moderno obliko, ki vsebuje tudi črne kamenčke, je igra dobila šele v 90. številki, kjer so jo tudi preimenovali v *Shiroshinju Kuroshinju*, kar pomeni "beli in črni biseri". Dodatek črnih kamenčkov je igri dodal globino in poskrbel za večjo popularnost. Moderno ime Masyu je igra dobila bolj za šalo, saj naj bi šlo za napačno izgovorjavo pismenke "biseri".

3.1.2 Pravila

Igra se dogaja na pravokotni mreži, na kateri so postavljeni beli in črni kamenčki. Cilj igre je na mreži narisati sklenjeno črto, ki poteka skozi središča



Slika 3.2: Vsa možna prečkanja kamenčkov, ki jih lahko tudi poljubno rotiramo

celic. Primer uganke in njeno rešitev lahko vidimo na sliki 3.1. Pravila igre so sledeča:

- Črta mora potekati skozi vsak kamenček.
- Črta mora skozi bel kamenček potovati naravnost, pred vhodom v kamenček ali po izhodu iz kamenčka pa zaviti. Zavije lahko tudi pri obeh (vse možnosti vidimo na sliki 3.2).
- Črta mora skozi črn kamenček obvezno zaviti, pred vhodom v kamenček in po izhodu iz kamenčka pa mora biti ravna (zadnji primer na sliki 3.2).

3.1.3 Dokaz NP-polnosti

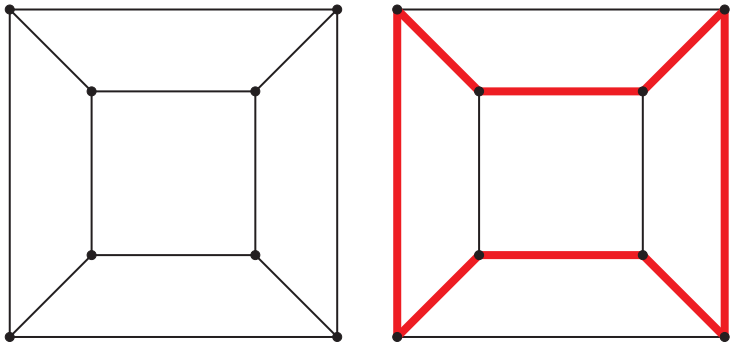
Problem 3. *Ali za dan primerek uganke Masyu obstaja rešitev?*

Najprej dokažemo, da je odločitveni problem Masyu v NP. Tega se lotimo tako, da za poljubno rešitev preverimo, če je pravilna. Preverjanje rešitve je sestavljeno iz dveh delov: preveriti moramo, da gre črta skozi vse kamenčke na pravilen način in je brez presečišč. Za prvi del porabimo kvečjemu $O(n^2)$ časa, saj ima uganka kvečjemu n^2 kamenčkov. Za drugi del porabimo $O(n^4)$ časa, ker ima uganka lahko kvečjemu n^2 povezav in zadostuje, da preverimo

vse pare povezav. Iz tega sledi, da je odločitveni problem Masyu v NP.

Da bi dokazali NP-polnost problema, bomo obstoječ NP-poln problem prevedli na Masyu. Obstoječ NP-poln problem bo v tem primeru iskanje Hamiltonskega cikla v planarnih kubičnih grafih (tak pristop je bil uporabljen že v [9]).

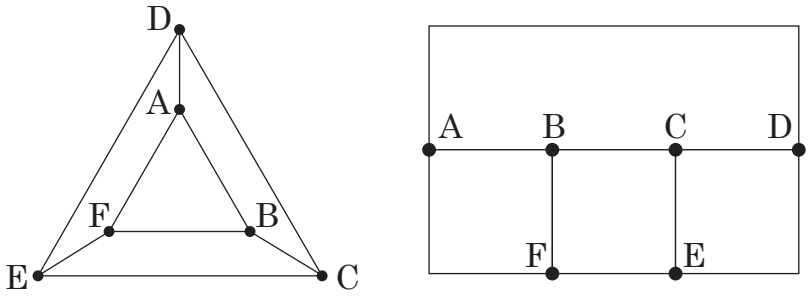
Problem 4. *Ali poljubni kubični planarni graf G vsebuje sklenjeno pot, ki vsako vozlišče obišče natanko enkrat?*



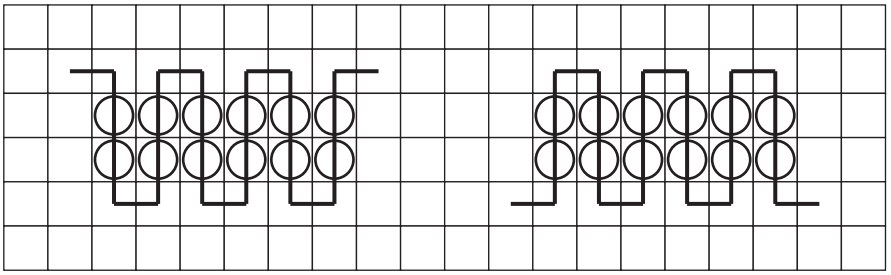
Slika 3.3: Primer ravnineskega kubičnega grafa (levo) in enak graf z vrisanim Hamiltonskim ciklom (desno)

Problem iskanja Hamiltonskih ciklov v kubičnih planarnih grafih je NP-poln [10], primer Hamiltonskega cikla pa lahko vidimo na sliki 3.3. Da bi pokazali NP-polnost problema Masyu, bomo opisali način izgradnje Masyu ugank, ki bodo ponazarjale poljubne kubične planarne grafe. Pri izgradnji bomo uporabljali zgolj bele kamenčke. Ta razred Masyu ugank bo imel rešitev natanko tedaj, ko bodo ustrezni kubični planarni grafi vsebovali Hamiltonski cikel.

Vhodne grafe bomo ponazorilli premočrtno, kar pomeni da bodo vse povezave v grafu bodisi pravokotne bodisi vzporedne ena drugi (slika 3.4). Vse planarne grafe je mogoče premočrtno ponazoriti [11].



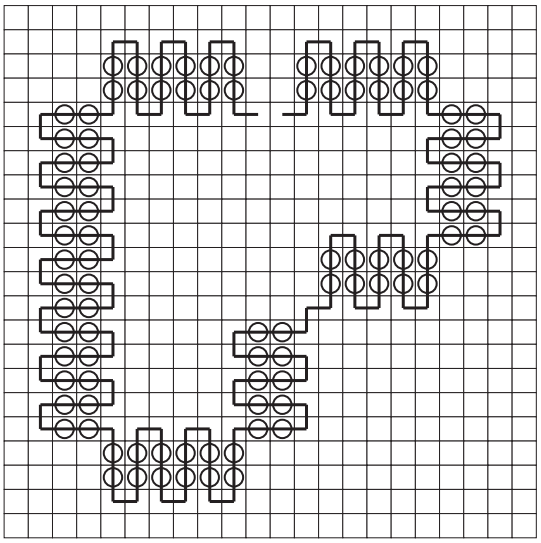
Slika 3.4: Primer planarnega grafa (levo) in njegova premočrtna ponazoritev (desno) [9]



Slika 3.5: Dve možni lokalni rešitvi stene iz belih kamenčkov [9]

Za vsako komponento planarnega kubičnega grafa moramo opisati pripadajočo strukturo Masyu uganke. Najprej bomo opisali, na kakšen način predstavimo povezave kubičnega planarnega grafa. Z belimi kamenčki sestavljamo t. i. *stene* debeline 2 poljubne dolžine. Takšna struktura nam zagotavlja, da črta poteka skozi stene na točno določen način: črta mora ob prebitju stene takoj zaviti in se usmeriti nazaj v steno (slika 3.5). Na tak način zasnovana stena se lahko tudi pravokotno poveže z drugimi stenami. Tako lahko sestavljamo tudi bolj zapletene strukture (slika 3.6), ki nam bodo prišle prav nekoliko kasneje.

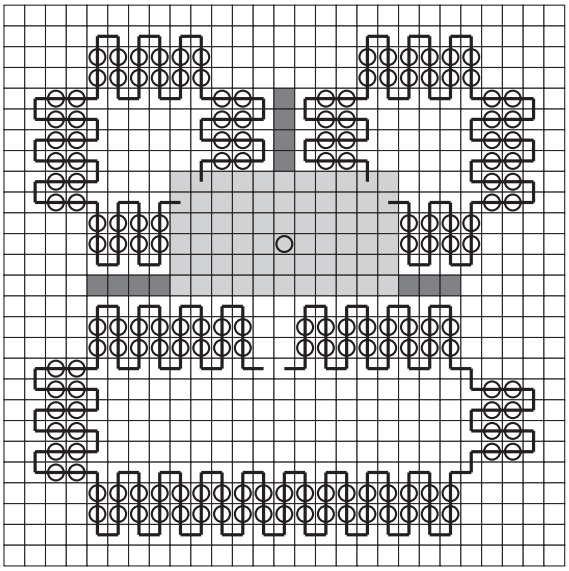
Med stenami tečejo t. i. *prehodi*, ki so nizi celic debeline 3 (pri čemer sta



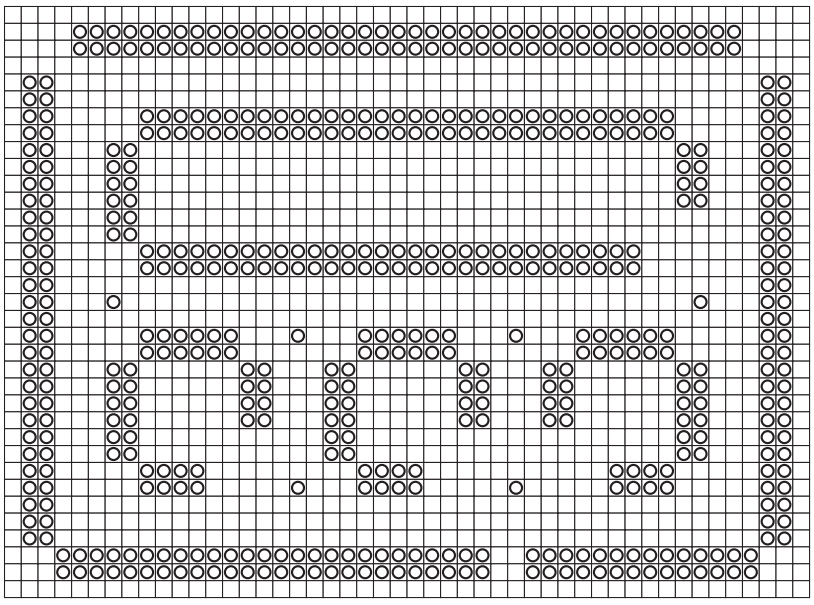
Slika 3.6: Iz sten lahko sestavimo kompleksne strukture [9]

zunanji dve celici zagotovo del Masyu poti zaradi stene iz belih kamenčkov) in poljubne dolžine. Vse povezave kubičnega planarnega grafa ponazorimo s prehodi.

Na vsakem vogalu ali preseku prehodov se nahajajo skupine praznih celic ali t. i. *sobane*. Slednje lahko zgradimo s prej omenjenim pravokotnim povezovanjem sten. V izbrane sobane bomo vstavili en sam bel kamenček, ki se ne bo smel dotikati nobenih drugih kamenčkov (slika 3.7). Vsa vozlišča planarnega kubičnega grafa ponazorimo s sobanami, v katere smo vstavili bel kamenček.



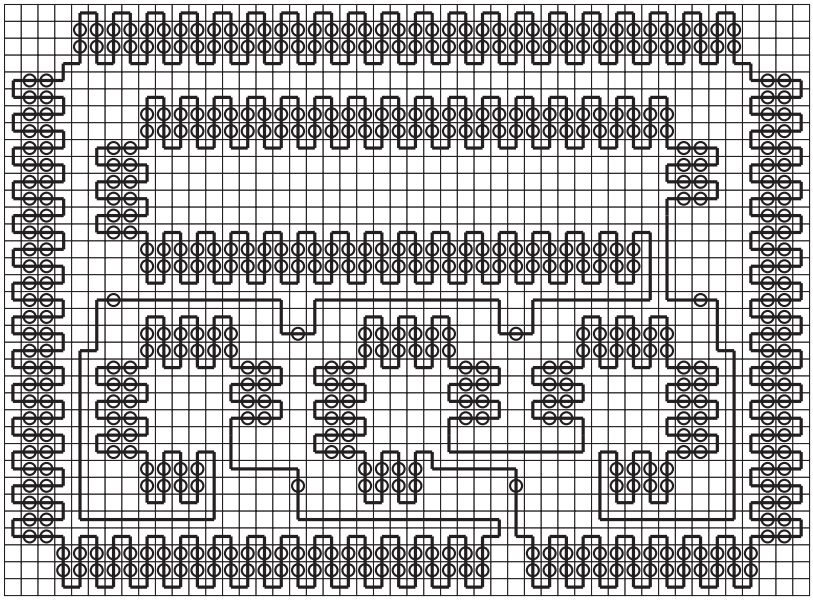
Slika 3.7: Svetlo sivo pobarvan del predstavlja sobano, temno sivi pa so prehodi [9]



Slika 3.8: Prazna uganka, ki predstavlja kubični planarni graf, ki ga najdemo na sliki 3.4 [9]

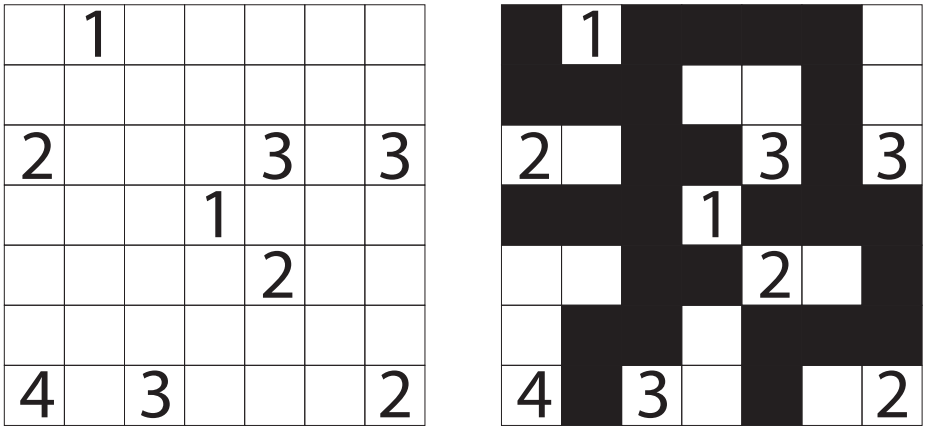
S tem dosežemo, da bo rešitev uganke Masyu tako izbrane sobane obiskala natanko enkrat. Ker je izbrani planarni graf kubičen (torej ima vsako vozlišče stopnjo 3) in so prehodi debeli 1 celico (ko vštejemo pot skozi stene), lahko vsako sobano obiščemo natanko enkrat, saj porabimo en prehod za prihod in enega za odhod (tretji prehod pa ostane prazen). Krajišči prehoda, ki ga omejuje vsaka posamezna stena, morata biti v sobani, ki predstavlja vozlišče pripadajočega grafa, s čimer zagotovimo, da bo vsaka stena del rešitve, ko obiščemo pripadajočo sobano. Tako sestavljene Masyu uganke imajo rešitev natanko tedaj, ko imajo pripadajoči grafi Hamiltonski cikel (sliki 3.8 in 3.9).

Iz zgornje konstrukcije Masyu ugank in vsebovanosti v NP sledi, da je uganka Masyu NP-polna.



Slika 3.9: Rešitev uganke, ki predstavlja določen kubični planarni graf, ki ga najdemo na sliki 3.4 [9]

3.2 Nurikabe



Slika 3.10: Na levi vidimo začetno stanje Nurikabe uganke, na desni pa podano rešitev [6].

3.2.1 Podatki o igri

Nurikabe (angl. *Coating Wall*) je miselna igra, ki jo je razvilo podjetje Nikoli. Ime izhaja iz japonske folklore in predstavlja duha, ki lahko zavzame obliko nevidnega zidu in ponoči preprečuje prehod popotnikom. Igra se je prvič pojavila v 33. številki revije *Puzzle Communication Nikoli* in hitro postala izjemno popularna.

3.2.2 Pravila

Igra se dogaja na mreži velikosti $n \times n$, ki jo sestavljajo bele celice, ki so bodisi prazne bodisi vsebujejo naravna števila. Cilj igre je celice pobarvati, oziroma jih pustiti pri miru, pri čemer morajo veljati naslednja pravila:

- Celic, ki vsebujejo števila, ne smemo pobarvati.

- Naravno število v celici nam pove velikost t. i. *stene*, ki je sestavljena iz belih celic, ki so med seboj povezane horizontalno ali vertikalno (diagonalne povezave ne štejejo). Vsaka stena vsebuje natanko eno naravno število in je horizontalno in vertikalno omejena s pobarvanimi celicami. Celica, ki vsebuje naravno število, je del stene.

- Vse pobarvane celice morajo tvoriti t. i. *potok*, ki je sklenjeno območje (med vsakim parom pobarvanih celic obstaja premočrtna pot pobarvanih celic).

- Pobarvane oziroma črne celice skupaj ne smejo tvoriti t. i. *bazena*, območja velikosti 2×2.

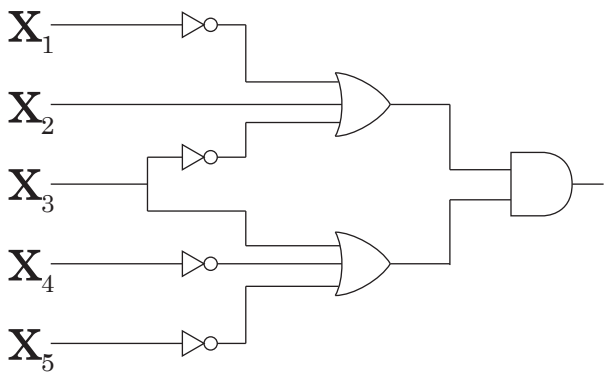
3.2.3 Dokaz NP-polnosti

Problem 5. *Ali za dan primerek uganke Nurikabe obstaja rešitev?*

Da bi dokazali NP-polnost problema, bomo obstoječ NP-poln problem planarni 3SAT prevedli na Nurikabe (tak pristop je bil uporabljen v [6]). Planarni 3SAT je po obliki podoben problemu SAT, razlikuje se le v zagotovljeni planarnosti in velikosti stavkov C; pri SAT imajo stavki lahko poljubno velikost, pri planarnem 3SAT pa imajo vedno velikost 3.

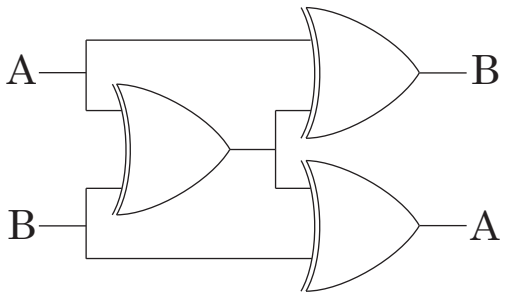
Problem 6. *Če imamo množico spremenljivk* $X = \{x_1, ..., x_n\}$ *in množico stavkov* $C = \{c_1, ..., c_m\}$ *nad* X*, pri čemer velja* $\forall c \in C; |c| = 3$ *in je dvodelni graf* $G = (X \cup C, E)$ *planaren, kjer je* $E = \{(x_i, c_j) | x_i \in c_j \text{ ali } \overline{x_i} \in c_j\}$*, ali obstaja izpolnljiva resničnostna nastavitev za* C*?*

NP-polnost planarnega 3SAT je pokazana v [7]. Omenjeni graf G lahko brez težav pretvorimo v logično vezje. To storimo tako, da vsa vozlišča spremenljivk zamenjamo z vhodi, vozlišča stavkov z logičnimi ALI-vrati (angl. OR-gate) oziroma kaskado ALI vrat. Za zaključek na vseh mestih, kjer je to potrebno, namestimo še negacije, oziroma NE-vrata (angl. NOT-gate). Negacije očitno nameščamo samo med vhodi in ALI-vrati glede na obliko



Slika 3.11: Planarno 3SAT vezje, sestavljeno iz NE-, ALI- ter IN-vrat, ki ustreza izjavi $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_4 \vee \neg x_5)$

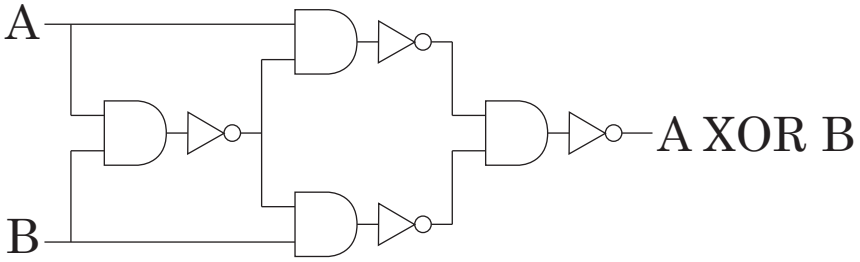
izbranega stavka. Izhod vseh ALI-vrat povežemo z logičnimi IN-vrati. Izhod vseh ALI-vrat in posledično IN-vrat so resnična natanko tedaj, ko je 3SAT izpolnljiv (slika 3.11).



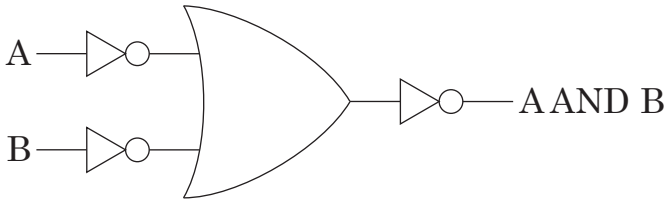
Slika 3.12: Prekrivanje žic lahko realiziramo z izključujočimi ALI-vrati

Očitno je tako nastalo vezje planarno natanko tedaj, ko je planaren graf G. Na tem mestu tudi opazimo, da planarnost ni nujen pogoj za dokaz NP-polnosti, saj bi lahko prekrivanje žic ponazorili s tremi izključujočimi ALI-vrati (slika 3.12). Izključujoča ALI-vrata lahko mimogrede sestavimo s kombinacijo IN- in NE- vrat (slika 3.13). Prav tako velja omeniti, da nam niti tukaj, niti kje drugje v tako zasnovanih vezjih ni treba eksplicitno upo-

rabljati IN-vrat, saj jih lahko po DeMorganovem pravilu[1] vedno sestavimo iz ALI- in NE-vrat (slika 3.14).



Slika 3.13: Izključujoča ALI-vrata, ki so sestavljena s štirimi negacijami ter IN-vrati



Slika 3.14: IN-vrata, ki so sestavljena iz treh negacij in ALI-vrat

Pogoj planarnosti smo si izbrali zgolj zato, ker je sam dokaz NP-polnosti zaradi tega krajši. Logično vezje z IN-, ALI- in NE- vrati lahko zgradimo iz naslednjih komponent:

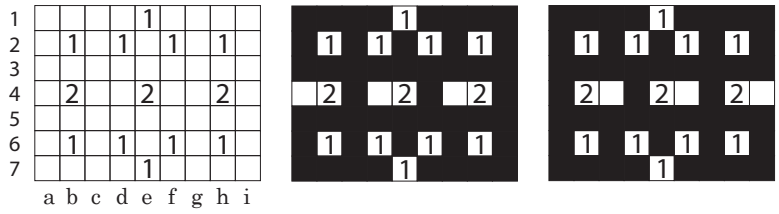
- žic, ki lahko prenašajo bodisi resnično bodisi neresično vrednost,
- vhodov in izhodov,
- razdelilcev žičnega signala,
- NE-vrat,
- ALI-vrat,

[1] $\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$

- (IN-vrat ne potrebujemo, saj jih lahko po že omenjenem DeMorganovem pravilu zgradimo iz kombinacije ALI- in NE-vrat),

- (Potrebno bo ustvariti tudi vrata s faznim zamikom zaradi problema, ki je omenjen spodaj).

Vsako od zgornjih komponent bomo zgradili s pomočjo pod-uganke Nurikabe. V spodnjih opisih pod-ugnank bomo za označbo posameznih celic uporabljali notacijo, kjer bomo pozicijo celice na vertikalni osi označili s številko, pozicijo na horizontalni osi pa s črko.

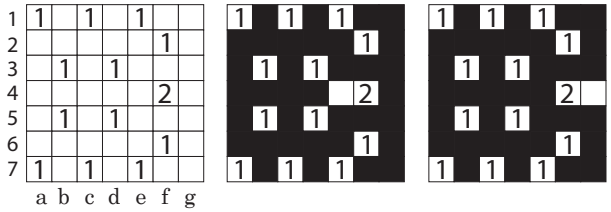


Slika 3.15: Prazna pod-uganka, ki ponazarja ravno žico, ki nosi signal od leve proti desni (levo), rešitev z resnično informacijo (sredina) in rešitev z neresnično informacijo (desno) [6]

Najprej bomo s pod-uganko ponazorili žico vezja. Na sliki 3.15 lahko vidimo primer žice, ki nosi informacijo od leve proti desni strani. Takoj opazimo, da morajo imeti celice s številko 2 nepobarvano polje bodisi levo bodisi desno od sebe, nikoli pa ni mogoče, da ga imajo neposredno nad oziroma pod seboj, ker ga blokirajo celice s številko 1, ali pa tako barvanje ustvari nepovezano črno področje (celica $e6$ ali $e2$). Iz tega sledi, da za pod-uganko obstajata natanko dve rešitvi, ki ponazarjata resnično in neresnično informacijo.

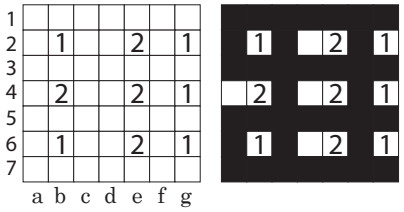
Rečemo, da žica nosi resnično informacijo, ko se belo polje nahaja za poljem s številko 2 glede na smer prenosa signala, in neresnično informacijo,

ko se belo polje nahaja pred poljem s številko 2 glede na smer prenosa signala.



Slika 3.16: Prazna pod-uganka, ki predstavlja vhod (levo), rešitev z resnično vrednostjo (sredina) in rešitev z neresnično vrednostjo (desno). Signal se prenaša od leve proti desni [6].

S prilagoditvijo pod-uganke, ki ponazarja ravno žico, lahko ponazorimo tudi vhod (slika 3.16) in izhod (slika 3.17). Vhod ima dve možni rešitvi, ki ustrezata resnični in neresnični vrednosti, izhod pa ima samo eno rešitev, kar ponazarja preverjanje ali žica nosi resnično informacijo.



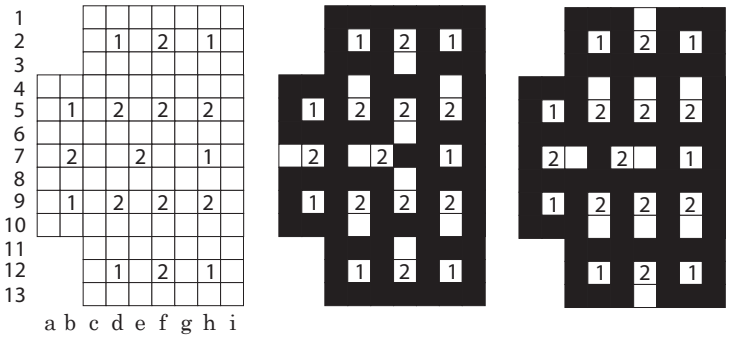
Slika 3.17: Prazna pod-uganka, ki predstavlja izhod (levo) in rešitev, ki preverja ali signal nosi resnično vrednost. Signal se prenaša od leve proti desni [6].

Naslednja pod-uganka prikazuje delitev žice in posredno signala (slika 3.18). Signal je predstavljen na isti način kot pri ravni žici. Zaradi podobne strukture pod-uganke ima tudi ta natanko dve rešitvi:

- Polje $a7$ je belo, kar pomeni, da je belo tudi polje $d7$. Če si nadalje ogledamo področji velikosti 2×2, sestavljeni iz $f7, f6, g7, g6$ in

$f8, f7, g8, g7$, opazimo, da mora biti vsaj eno od omenjenih polj belo. Iz tega sledi, da morata biti polji $f6$ in $f8$ beli, nadalje pa morata biti beli tudi polji $f3$ in $f11$.

- Polje $c7$ je belo, kar pomeni, da je belo tudi polje $f7$. Edini način, da ustrezno razporedimo bela polja ob celicah s številko 2, je izbira $f4$ in $f10$, kar pa vpliva na polji $f1$ in $f13$, ki morata prav tako postati beli.

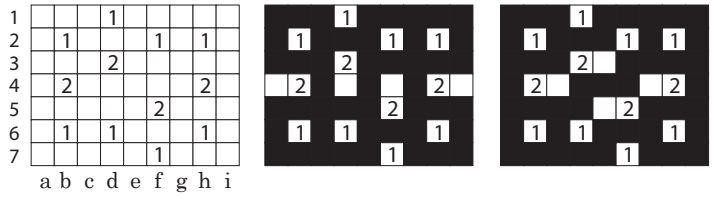


Slika 3.18: Prazna pod-uganka, ki predstavlja delitev žice (levo), rešitev z resnično vrednostjo (sredina) in rešitev z neresnično vrednostjo (desno). Signal prihaja z leve strani in se deli navzgor in navzdol [6].

Zgornji dve možnosti sta edina dva načina, da rešimo pod-uganko, ki predstavlja delitev žice. Velja omeniti, da se pod-uganko lahko uporabi tudi za preprosto lomljenje žice, pri čemer pa moramo enega od dveh novo nastalih koncev terminirati s pod-uganko s slike 3.16.

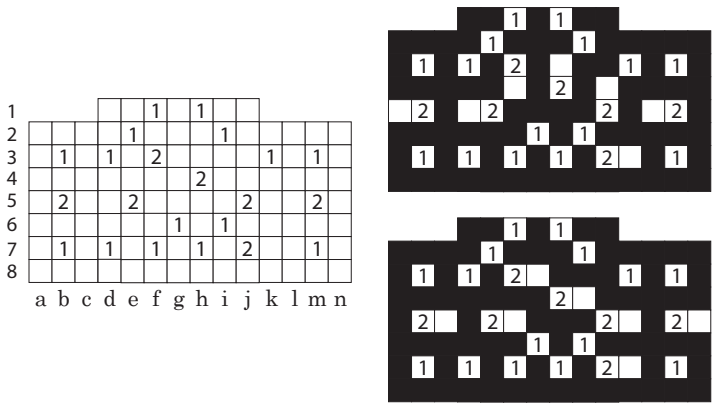
Sledi opis logičnih vrat. Pod-uganko, ki predstavlja NE-vrata, lahko vidimo na sliki 3.19. Tudi ta pod-uganka ima natanko dve možni rešitvi:

- Polje $a4$ je belo. Področje velikosti 2×2, sestavljeno iz polj $c4$, $c5$, $d4$, $d5$, mora vsebovati vsaj eno belo polje. To lahko dosežemo samo tako, da pustimo polje $d4$ belo. Na podoben način sledi, da je izmed polj $e3$, $e4$, $f3$, $f4$, belo polje $f4$. Iz tega sledi, da je polje $i4$ belo. Na tak način spremenimo vrednost signala iz resnične v neresnično.



Slika 3.19: Prazna pod-uganke, ki predstavlja NE-vrata (levo), rešitev z resnično vrednostjo (sredina) in rešitev z neresnično vrednostjo (desno) [6]

- Polje $c4$ je belo. Edini način, da razporedimo belo polje ob celici $d3$ (ki ima številko 2) je, da izberemo $e3$. Tudi tukaj opazimo, da mora biti od polj $d4$, $d5$, $e4$, $e5$ vsaj eno polje belo in da je to polje $e5$, zaradi tega pa postane belo tudi polje $g4$. Taka rešitev spremeni vrednost signala iz neresnične v resnično.



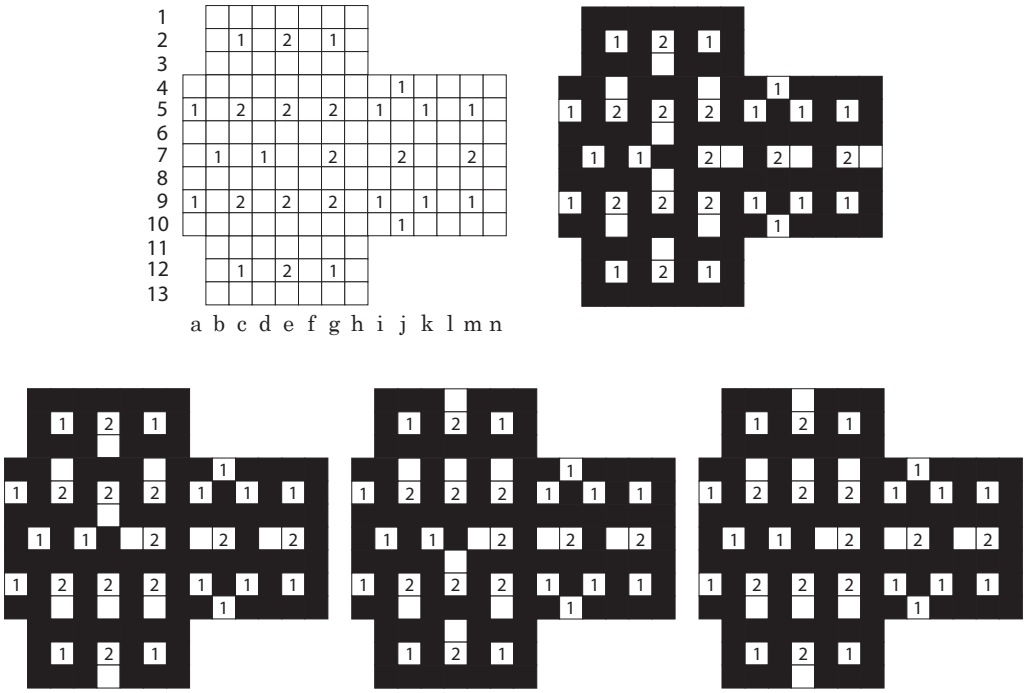
Slika 3.20: Prazna pod-uganka, ki predstavlja vrata s faznim zamikom (levo), rešitev z resnično vrednostjo (desno zgoraj) in rešitev z neresnično vrednostjo (desno spodaj) [6]

Zgornji dve možnosti sta edina dva načina, da rešimo pod-uganko, ki predstavlja NE-vrata.

Sledi opis ALI-vrat, tukaj pa nastopi problem. Signal v vezju ponazorimo z zaporedjem polj s številko 2, tak pristop pa privzame, da v vhod ALI-vrat

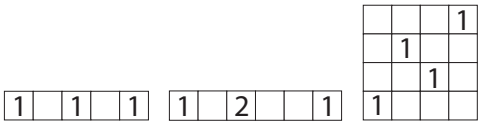
prispeta oba signala žic z isto fazo. To na žalost ni vedno res, ker lomljenje žice izniči periodičnost polj s številko 2 in zamakne fazo. Zaradi tega problema moramo zasnovati tudi pod-uganko, ki predstavlja vrata s faznim zamikom (slika 3.20). Rešitvi faznega zamika izpeljemo podobno kot NE-vrata.

Sedaj se lahko lotimo opisa ALI-vrat (slika 3.21). Za razumevanje ALI-vrat se moramo posebej osredotočiti na polje $e7$.



Slika 3.21: Prazna uganka, ki predstavlja ALI-vrata (zgoraj levo), rešitev z zgornjo in spodnjo neresnično vrednostjo (zgoraj desno), rešitev z zgornjo neresnično in spodnjo resnično vrednostjo (spodaj levo), rešitev z zgornjo resnično in spodnjo neresnično vrednostjo (spodaj sredina) in rešitev z zgornjo in spodnjo resnično vrednostjo (spodaj desno) [6].

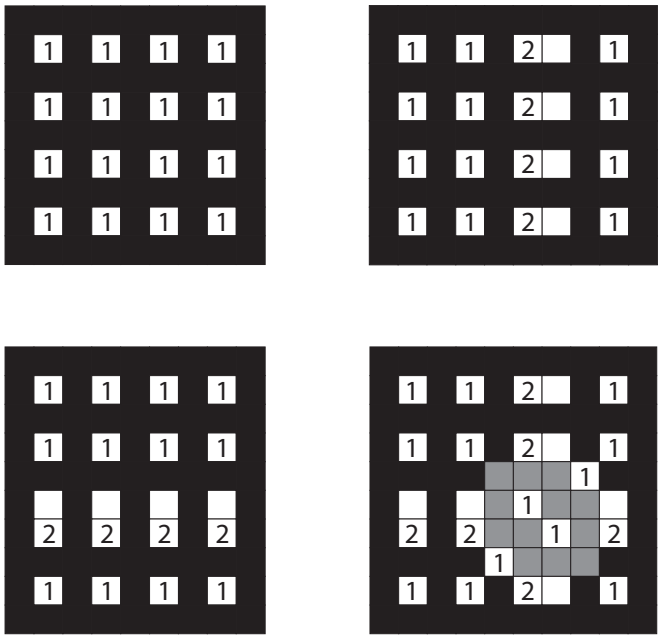
V vseh primerih, kjer je tako polje $e6$ kot tudi polje $e8$ belo (takrat gre za podaljška otokov, ki izhajata iz polj $e5$ in $e9$), mora biti polje $f7$ gotovo pobarvano, kajti sicer je polje $e7$, ki mora biti pobarvano v vsakem primeru, saj je sosednje polju s številko 1, odrezano od potoka. Iz tega sledi, da morajo biti polja $h7$, $k7$ in $n7$ bela. To pa pomeni, da je ob dveh vhodnih neresničnih vrednostih tudi izhod neresnična vrednost. Če je vsaj ena od vhodnih vrednosti resnična, pomeni da je posledično pobarvano vsaj eno od polj $e8$ in $e6$. Iz tega sledi, da mora biti polje $f7$ obvezno belo, saj sicer dobimo pobarvano področje velikosti 2×2 bodisi iz polj $e6$, $f6$, $e7$ in $f7$ bodisi iz polj $e7$, $f7$, $e8$ in $f8$. Iz tega sledi, da sta tudi polji $i7$ in $l7$ beli. To pa pomeni, da vse kombinacije vhodnih vrednosti, od katerih je vsaj ena resnična, vrnejo resnično vrednost.



Slika 3.22: Liha vrstica (levo) in soda vrstica (sredina), zapolnjeni po opisanem algoritmu. Desno lahko vidimo, kako razširjeni algoritem zapolne področje velikosti 4×4 [6].

Da bi dobili optimalen rezultat, moramo prav tako določiti, kaj storimo s celicami, ki ne predstavljajo noben del vezja (to področje bomo v nadaljevanju imenovali "nepokrito področje"). Ta problem rešimo z ustrezno postavitvijo številk 1 in 2, tako da bo nastali potok vedno povezan in brez bazenov. Predstavljamo si, da je nepokrito področje razdeljeno na različne pravokotnike, ki so zapolnjeni od zgoraj navzdol vsako vrstico posebej. Med posameznimi polji s števili so seveda prazna polja. Ker lahko prilagajamo dolžino žic, lahko zagotovimo, da je velikost pravokotnikov, ki jih sestavljajo prazna polja, vedno vsaj 5×5 (če ne štejemo zunanjih pobarvanih polj).
Najprej pokažemo, kako rekurzivno sestavimo eno vrstico: če je dolžina vrstice bodisi 1 bodisi 2, potem prvo polje označimo z bodisi 1 bodisi 2 in

zaključimo. Sicer označimo prvo polje z 1, odrežemo prvi dve polji vrstice in pogledamo, če je dolžina preostanka 1 ali 2. Če je, potem rekurzivno nadaljujemo s preostankom. Sicer pa označimo zadnje polje z 1, odrežemo zadnji dve polji vrstice in rekurzivno nadaljujemo s preostankom. To nam da dva možna vzorca, ki ju lahko vidimo na sliki 3.22.



Slika 3.23: Pravokotniki z dimenzijami: liha × liha (zgoraj levo), liha × soda (zgoraj desno), soda × liha (spodaj levo) in soda × soda (spodaj desno). V zadnjem primeru tudi vidimo polje velikosti 4×4, ki je zapolnjeno na poseben način [6].

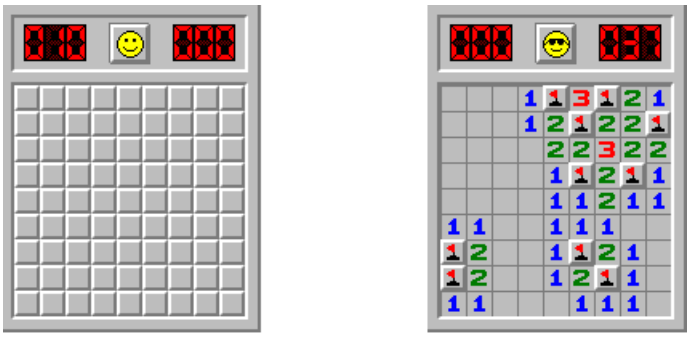
Pravokotnik zapolnimo na sledeč način: prvo vrstico in prvi stolpec zapolnimo z zgornjim algoritmom. Nato vse nadaljnje vrstice in stolpce skušamo zapolniti z vzorcem iz prve vrstice oziroma prvega stolpca (seveda se vmes nahajajo prazne vrstice). Tak pristop je uspešen v vseh primerih, razen ko sta tako višina kot širina pravokotnika sodi števili. V tem primeru nam ostane področje velikosti 4×4, ki ga zapolnimo z vzorcem s slike 3.22. Primere

zapolnitev različnih pravokotnikov lahko vidimo na sliki 3.23.

Z zgornjimi pod-ugankami lahko sestavimo poljubno planarno 3SAT vezje, obenem pa velja, da je tako sestavljeno planarno 3SAT vezje izpolnljivo natanko tedaj, ko je Nurikabe uganka rešljiva. Isti pristop je bil uporabljen v [6].

Vsebovanost v NP je očitna, saj lahko za dano rešitev pobarvanih celic v polinomskem času ugotovimo, ali je pravilna (preverimo sklenjenost potoka, velikost in omejenost sten ter neobstoj bazenov). Iz tega in zgornje prevedbe sledi, da je odločitveni problem Nurikabe NP-poln.

3.3 Minolovec



Slika 3.24: Začetno stanje igrice Minolovec (levo) in uspešno rešena igrica (desno).

3.3.1 Podatki o igri

Minolovec (angl. *Minesweeper*) je računalniška igrica, ki je uporabnikom dostopna kot dodatek operacijskega sistema *Microsoft Windows*. Prvič je izšla leta 1990 kot del *Microsoft Entertainment Pack 1*, od takrat pa je bila del vsakega *Windows* paketa. V paketu *Windows Vista* je igrica dobila prenovljen izgled, prav tako pa je igralcem nekoliko olajšala igranje, saj v prvi potezi nikoli niso naleteli na mino. Igrica ni bila del paketa *Windows 8*, vseeno pa so jo igralci lahko dobili preko platforme *Windows Store*.

3.3.2 Pravila

Igrica se igra na mreži velikosti $n \times n$, ki jo sestavljajo zakrita polja. Polja postopoma odkrivamo, na njih pa se lahko nahajajo mine. V primeru, da odkrijemo mino, ta detonira, igra pa se zaključi. Odkrita polja brez min vsebujejo število od 0 do 8. To število nam pove, koliko min se nahaja v neposredni bližini odkritega polja (torej levo, desno, gor, dol in neposredno v štirih diagonalnih smereh). To informacijo uporabimo tako, da polja z minami posebej označimo. Cilj igre je označiti vsa polja, na katerih se nahajajo mine. Prav tako ne smemo odkriti nobenega polja, na katerem se nahaja

mina. Primer delno rešene uganke lahko vidimo na sliki 3.25.

		2	1	2	1
		3	*	4	
2	2	3	*	5	
0	0	1	1	4	
0	1	1	1	2	
0	1				

Slika 3.25: Delno rešena mreža igrice Minolovec. Polja s številkami nam povejo, koliko min se nahaja v neposredni bližini, polja z zvezdico imajo mino, neodkrita polja pa so prazna [8].

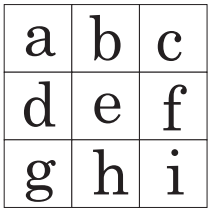
Posplošena uganka (ali problem) minolovca je nekoliko drugačna, vseeno pa se ohranja bistvo igre. Začetno stanje ni mreža z zakritimi polji, temveč mreža, kjer je določeno število polj že odkritih. Tako kot pri igrici tudi tukaj odkrita polja vsebujejo številke, cilj uganke pa je na neodkrita polja postaviti mine na tak način, da je postavitev min in številk smiselna ali *konsistentna*. Za tako zastavljeno uganko lahko rešujemo odločitveni problem 7.

3.3.3 Dokaz NP-polnosti

Problem 7. *Ali za dano mrežo iz polj, ki so bodisi zakrita bodisi označena z mino bodisi odkrita in vsebujejo število od* 0 *do* 8*, obstaja razporeditev min na zakrita polja, tako da so odkrita števila ustrezna?*

Vsebovanost v NP je očitna, saj lahko za dano razporeditev min v polinomskem času ugotovimo, ali je pravilna (pregledamo okolice vseh odkritih polj s števili).

Posplošeni problem Minolovca lahko dokaj preprosto prevedemo na SAT, saj lahko pravila igre predstavimo z logičnim vezjem na sledeč način: denimo,



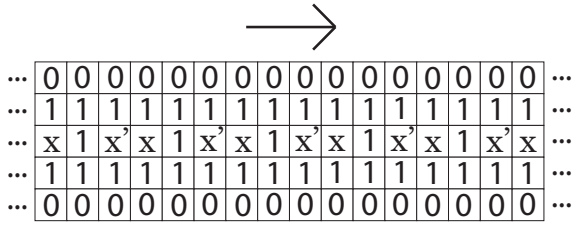
Slika 3.26: Poljubna mreža velikosti 3×3, ki jo uporabimo za prikaz prevedbe Minolovca na SAT (ta smer dokazovanja ni zadostna za dokaz NP-polnosti) [8].

da imamo podano mrežo velikosti 3×3, kot jo vidimo na sliki 3.26. Naj spremenljivka a_m pomeni "na polju a se nahaja mina" in naj pri $0 \leq j \leq 8$ a_j pomeni "na polju a in na natanko j sosednjih poljih ni mine". Isto naj velja za $b, c, d, ..., i$. Potem lahko pravila Minolovca za središčno polje e opišemo s spodnjimi izjavami:

- Natanko ena od spremenljivk $e_m, e_0, e_1, ..., e_8$ je resnična.
- Pri $k = 0, 1, ..., 8$ velja, če je e_k resnična, potem je natanko k spremenljivk od $a_m, b_m, c_m, d_m, f_m, g_m, h_m, i_m$ resničnih.

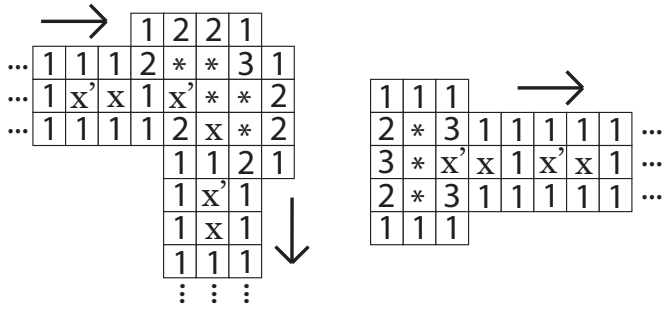
Zgornje izjave lahko izrazimo z logičnimi vezji, ki za vhod sprejmejo a_m, a_0, ..., a_8, b_m, b_0, ..., b_8, ..., i_8 (skupno 90 vhodov). Če s C označimo vezje, ki ga sestavljajo tako sestavljena vezja za vsako posamezno polje in vse izhode združimo v IN-vrata, potem postane izbrani problem Minolovca ekvivalenten primerku problema SAT. To seveda ni dovolj za dokaz NP-polnosti, saj moramo prikazati prevedbo v drugo smer. Postopek dokazovanja je podoben kot pri uganki Nurikabe, saj moramo zgraditi pod-uganke, ki predstavljajo različne dele (v tem primeru SAT) vezja.

Najprej moramo zasnovati žico vezja, ki bo nosila signal (slika 3.27). Na sliki smo z x in x' označili polja, ki bodisi imajo bodisi nimajo mine. Opazimo, da ima takšna zasnova žice samo dve možni rešitvi: bodisi imajo mino vsa x polja (v tem primeru žica nosi resnično vrednost) bodisi vsa x' polja



Slika 3.27: Pod-uganka, ki predstavlja žico. Signal se prenaša od leve proti desni [8].

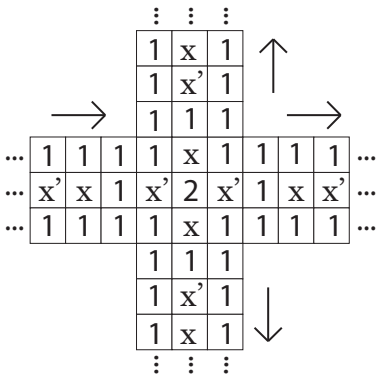
(v tem primeru žica nosi neresnično vrednost). Potrebno je omeniti, da sta x in x' v tem primeru zgolj pozicijski označbi. Žica v resnici nosi resnično vrednost, ko se mine nahajajo neposredno za sredinskimi polji s številko 1 glede na smer signala in neresnično vrednost, ko se mine nahajajo neposredno pred sredinskimi polji s številko 1 glede na smer signala.



Slika 3.28: Pod-uganka, ki predstavlja lomljenje žice (levo), in pod-uganka, ki predstavlja zaključitev žice [8]

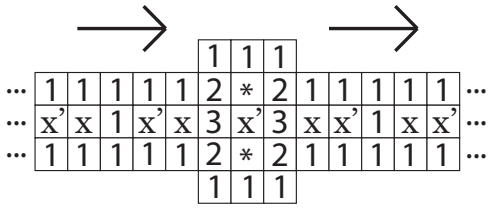
Tudi pri tej prevedbi moramo zagotoviti lomljenje in deljenje žic. Na sliki 3.28 lahko vidimo lomljenje žice, prav tako pa vidimo zasnovo za zaključitev žice. Slednjega bomo kasneje lahko uporabili tudi kot izhod, ki bo žico prisilil, da nosi bodisi resnično bodisi neresnično vrednost. Tako kot pri primeru s slike 3.25 imajo tudi tukaj polja, ki smo jih označili z zvezdico, mino. Označba sicer ni nujna, saj je s sosednjih polj razvidno, da se na poljih

res nahajajo mine. Na tem mestu tudi velja omeniti, da vsa polja, ki niso del vezja, označimo z 0 in jih tako izločimo iz reševanja problema.



Slika 3.29: Pod-uganka, ki predstavlja deljenje žice in posredno signala na tri dele. Srednji del signala se negira [8].

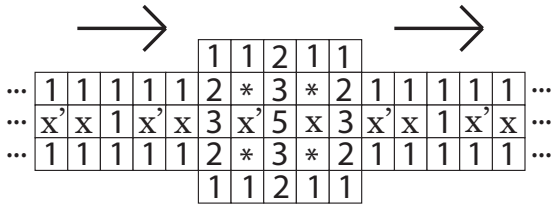
Slika 3.29 prikazuje deljenje žic. Kljub temu, da bi lahko NE-vrata zgradili s kombinacijo razdelilca žic in zaključitve, se bomo vseeno odločili za nekoliko preprostejšo predstavitev (slika 3.30). Preprostejša oblika nam razbremeni gradnjo samih NE-vrat, prav tako pa lahko dvoje NE-vrat združimo in tako na preprost način dobimo vrata s faznim zamikom. Slednja tudi tukaj rešujejo isti problem kot pri uganki Nurikabe (slika 3.31).



Slika 3.30: Pod-uganka, ki predstavlja NE-vrata. Opazimo, da se pojavitev min glede na sredinska polja s številko 1 spremeni [8].

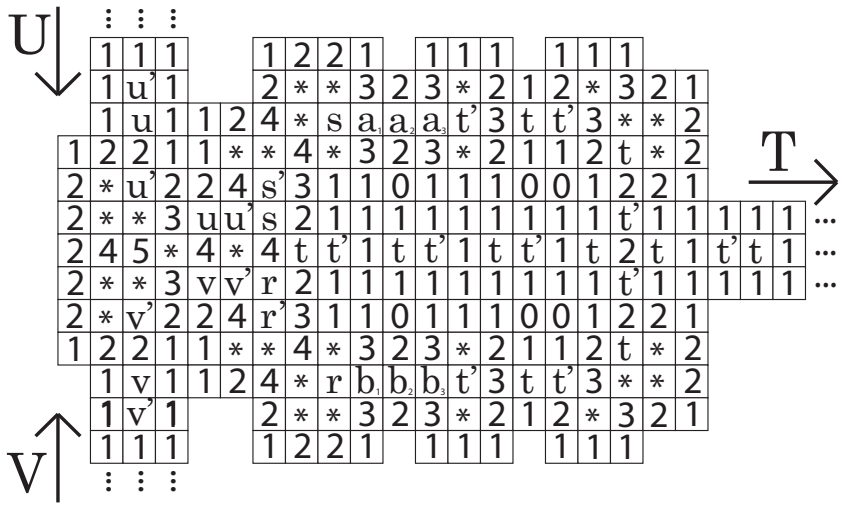
Za zaključek moramo zasnovati vsaj IN-vrata (namesto tega bi se seveda lahko odločili tudi za ALI-vrata). Mimogrede moramo tudi pokazati, da se

signali lahko križajo. Tukaj prav tako uporabimo strukturo s slike 3.12 in se spomnimo, da lahko izključujoča ALI-vrata sestavimo iz kombinacije IN- in NE-vrat (slika 3.13).



Slika 3.31: Pod-uganka, ki predstavlja vrata s faznim zamikom [8]

S tem znanjem lahko predstavimo IN-vrata (slika 3.32). Vrata so sestavljena iz dveh vhodnih žic U in V in imajo eno izhodno žico T. Prav tako vrata vsebujejo dve notranji žici R in S, ki se povežeta z delilnikom signala, katerega izhod je T. Strukturi, ki žici R in S povezujeta z žico T, označimo z a_1, a_2, a_3 in b_1, b_2, b_3.



Slika 3.32: Pod-uganka, ki predstavlja IN-vrata [8].

Najprej si oglejmo, kaj se zgodi v pod-uganki, če T nosi resnično vrednost. To pomeni, da so na poljih označenih s t, mine, polja označena s t', pa so prazna. Zaradi polja s številko 3 neposredno nad a_3 vemo, da morajo biti v okolici natanko 3 mine. Pozicijo ene od min že poznamo (polje z zvezdico), prav tako pa vemo, da je polje t' prazno, kar pomeni, da se mini gotovo nahajata na poljih a_2 in a_3. Nadalje opazimo, da ima polje s številko 2 neposredno nad a_2 v svoji okolici že 2 mini, kar pomeni, da je polje a_1 prazno. Zaradi polja s številko 3, ki se nahaja neposredno nad a_1, sedaj vemo, da mora imeti polje s mino. Na podoben način ima mino tudi polje r na drugi strani pod-uganke (tukaj seveda gledamo strukturo b_1, b_2, b_3). Osrednje polje s številko 4 (na sliki obarvano sivo) ima torej ob sebi že 4 mine: s, r, t in polje z zvezdico. To pomeni, da sta polji označeni z u' in v' prazni, posredno pa imajo polja u in v mine, kar pomeni, da obe žici U in V nosita resnično vrednost.

Sedaj si moramo ogledati še primer, kjer vsaj ena od žic U, V nosi neresnično vrednost. Ker želimo, da naša pod-uganka v tem primeru vrne neresnično vrednost, bodo imela polja označena s t' mino. Potemtakem ima osrednje polje ob sebi bodisi 2 bodisi 3 mine, ki se nahajajo na poljih u', v' in na polju z zvezdico. To pomeni, da se bodisi na enem od polj s in r bodisi na obeh nahaja mina. Denimo, da se na polju s nahaja mina. Vemo, da se na natanko dveh poljih izmed polj a_1, a_2, a_3 nahaja mina. Opazimo, da ima natanko eno od polj a_1 in a_2 mino (zaradi polja s številko 3 neposredno nad a_1), to pa pomeni, da se mina gotovo nahaja na polju a_3. Nadalje tudi ugotovimo, da je polje a_2 prazno, saj ima polje s številko 3 neposredno nad a_3 ob sebi že 3 mine. Takšna postavitev je očitno ustrezna. Na isti način pokažemo ustreznost postavitve za r, postavitev pa je ustrezna tudi takrat, ko imata obe polji mino. To pomeni, da smo uspešno zasnovali tudi IN-vrata.

Z zgoraj opisanimi pod-ugankami lahko SAT prevedemo na uganko Minolovec. Iz tega in vsebovanosti v NP sledi, da je Minolovec NP-poln.

3.4 Latinski kvadrati

1	2	3	4
2	4		
3			
4			

1	2	3	4
2	4	1	3
3	1	4	2
4	3	2	1

Slika 3.33: Primer delnega latinskega kvadrata velikosti 4×4 (levo) in njegova rešitev (desno).

Dopolnjevanje latinskih kvadratov se običajno ne smatra za miselno igro, vendar bomo odločitveni problem vseeno obravnavali, saj je tesno povezan z miselno igro Sudoku, katere NP-polnost bomo dokazali prav s pomočjo NP-polnosti dopolnjevanja latinskih kvadratov.

3.4.1 Podatki o igri

Izraz "latinski kvadrat" je prvi uporabil matematik Leonhard Euler, ki je za simbole znotraj mreže kvadrata uporabljal latinske črke [13], poznali pa so jih že arabski numerologi iz 13. stoletja, ki so jih imenovali *wafq majazi* [14]. Struktura latinskih kvadratov se je skozi leta pojavila tudi v različnih miselnih igrah, na primer *Sudoku* in *Kenken*.

3.4.2 Pravila

Latinski kvadrat je mreža velikosti $n \times n$, v kateri se nahaja n različnih simbolov. V vsaki vrstici in vsakem stolpcu se vsak simbol pojavi natanko enkrat.

Brez škode za splošnost bomo privzeli, da so omenjeni simboli števila od 1 do n (slika 3.33). Delni latinski kvadrati so latinski kvadrati, ki imajo določeno število celic praznih.

3.4.3 Dokaz NP-polnosti

Problem 8. *Ali lahko podani delni latinski kvadrat dopolnimo do popolnega latinskega kvadrata?*

1			
2			
3			
			4

Slika 3.34: Primer delnega latinskega kvadrata velikosti 4×4, ki ga ne moremo dopolniti do popolnega latinskega kvadrata.

Primer delnega latinskega kvadrata, ki ga ne moremo dopolniti do popolnega latinskega kvadrata lahko vidimo na sliki 3.34. Da bi dokazali NP-polnost problema, bomo obstoječ NP-poln problem — delitev tridelnega grafa na trikotnike — prevedli na dopolnjevanje latinskih kvadratov (tak pristop je bil uporabljen v [15]). NP-polnost delitve tridelnega grafa na trikotnike bomo privzeli, dokaže pa jo [16].

Problem 9. *Ali lahko tridelni graf G razdelimo na q disjunktnih množic $T_1, T_2, ..., T_q$, pri čemer vsaka množica vsebuje natanko 3 vozlišča, ki skupaj tvorijo trikotnik v G?*

Imamo torej graf G, ki ga sestavljajo particije $V_1 \cup V_2 \cup V_3$. Vozlišča iz V_1 so označena z $\{a_1, a_2, ..., a_x\}$, pri čemer velja $x = |V_1|$. Na podoben način so vozlišča iz V_2 označena z $\{b_1, b_2, ..., b_y\}$ in vozlišča iz V_3 z $\{c_1, c_2, ..., c_z\}$. Definirajmo še nekaj pojmov, ki nam bodo služili kot povezave med dopolnjevanjem latinskih kvadratov in delitvjo tridelnih grafov na trikotnike:

Definicija 1. *Defekt od P je graf $G(P)$, ki pripada delnemu latinskemu kvadratu P. $G(P)$ ima naslednjo množico vozlišč: $\{a_i|$vrstica i vsebuje prazno celico$\}\cup$ $\{b_j|$stolpec j vsebuje prazno celico$\}\cup\{c_k|$simbol k se pojavi v manj kot n celicah$\}$. Povezave v $G(P)$ imajo naslednje lastnosti:*

- *Če je celica (i, j) v P prazna, potem je (a_i, b_j) povezava.*
- *Če vrstica i ne vsebuje simbola k, potem je (a_i, c_k) povezava.*
- *Če stolpec j ne vsebuje simbola k, potem je (b_j, c_k) povezava.*

$G(P)$ ima delitev na trikotnike natanko tedaj, ko lahko P dopolnimo do popolnega latinskega kvadrata.

Definicija 2. *Latinsko ogrodje $LF(G; r, s, t)$ za graf G je mreža velikosti $r \times s$. Vsaka celica mreže je bodisi prazna, bodisi vsebuje vrednost iz $\{1, ..., t\}$. Vsaka vrstica in stolpec vsebujeta vsak simbol kvečjemu enkrat. Prav tako veljajo naslednje tri omejitve:*

- *Če G vsebuje povezavo (a_i, b_j), potem je celica (i, j) v latinskem ogrodju prazna.*
- *Če G vsebuje povezavo (a_i, c_k), potem vrstica i latinskega ogrodja ne vsebuje simbola k.*
- *Če G vsebuje povezavo (b_j, c_k), potem stolpec j latinskega ogrodja ne vsebuje simbola k.*

Ko velja $r = s = t$, je G natanko defekt od $LF(G; r, r, r)$. Velja tudi, da je $LF(G; r, r, r)$ delni latinski kvadrat, ki ga lahko dopolnimo do popolnega latinskega kvadrata natanko tedaj, ko lahko G delimo na trikotnike. Z uporabo lem iz [15] lahko postavimo sledeči izrek:

Izrek 3.4.1. *Za dani tridelni n-graf G lahko v polinomskem času zgradimo latinsko ogrodje $LF(G; 2n, 2n, 2n)$.*

Sedaj imamo dovolj znanja, da lahko dokažemo NP-polnost dopolnjevanja latinskih kvadratov.

Najprej moramo pokazati, da je dopolnjevanje latinskih kvadratov v NP. To sledi iz dejstva, da lahko poljubno rešitev preverimo v polinomskem času, saj moramo zgolj preveriti, da se v vsakem stolpcu in v vsaki vrstici nahajajo števila od 1 do n. Sedaj dopolnjevanje latinskih kvadratov prevedemo na delitev tridelnega grafa na trikotnike.

Za podani tridelni n-graf G (in podane particije) moramo najprej ugotoviti, ali so vse tri particije enake velikosti; če niso, potem ga ne moremo razdeliti na trikotnike. Če so, potem lahko v polinomskem času zgradimo latinsko ogrodje $LF(G; 2n, 2n, 2n)$, ki je delni latinski kvadrat. Tega lahko do popolnega latinskega kvadrata dopolnimo natanko tedaj, ko lahko G razdelimo na trikotnike. To namreč sledi iz tega, da je G defekt delnega latinskega kvadrata. Iz tega sledi, da je problem dopolnjevanja latinskih kvadratov NP-poln.

3.5 Sudoku

2			8		4			6
		6				5		
	7	4				9	2	
3				4				7
			3		5			
4				6				9
	1	9				7	4	
		8				2		
5			6		8			1

2	5	3	8	9	4	1	7	6
1	9	6	2	3	7	5	8	4
8	7	4	1	5	6	9	2	3
3	8	1	9	4	2	6	5	7
9	6	7	3	8	5	4	1	2
4	2	5	7	6	1	8	3	9
6	1	9	5	2	3	7	4	8
7	3	8	4	1	9	2	6	5
5	4	2	6	7	8	3	9	1

Slika 3.35: Primer klasične Sudoku uganke (levo) in rešitev uganke (desno).

3.5.1 Podatki o igri

Moderni Sudoku izvira iz New Yorka, kjer je bil leta 1979 objavljen v reviji *Dell Magazines* pod imenom *Number Place*, nato pa se je skokovito širil po celem svetu in med drugim prišel tudi na Japonsko, kjer je dobil danes bolj znano ime. Ime *sudoku* je kratica stavka v japonščini *suuji wa dokushin ni kagiru*, ki se približno prevede v: "Števila se lahko pojavijo samo enkrat".

3.5.2 Pravila

Klasični sudoku se dogaja na mreži velikosti 9×9 (slika 3.35), vendar bomo v spodnjem odstavku opisali posplošeno varianto.

Primerek igre Sudoku je mreža velikosti $n^2 \times n^2$. Mreža je dodatno razdeljena še na n^2 samostojnih *podmrež* velikosti $n \times n$. Mrežo sestavljajo celice, ki so prazne, ali pa je v njih številka od 1 do n^2. Cilj igre je s številkami zapolniti celotno mrežo, pri čemer morajo veljati naslednje omejitve:

- V vsaki vrstici morajo biti števila od 1 do n^2.

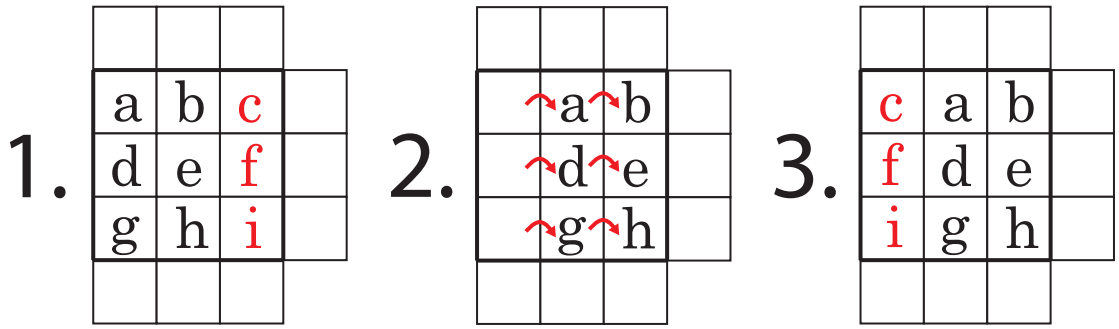
- V vsakem stolpcu morajo biti števila od 1 do n^2.
- V vsaki podmreži morajo biti števila od 1 do n^2.

3.5.3 Dokaz NP-polnosti

Problem 10. *Ali za podani primerek igre Sudoku obstaja rešitev?*

NP-polnost odločitvenega problema Sudoku je bila pokazana v [12]. Pokazati moramo, da se dopolnjevanje latinskih kvadratov lahko prevede na Sudoku. To dosežemo tako, da za podani delni latinski kvadrat velikosti $n \times n$ zgradimo pripadajoči primerek igre Sudoku velikosti $n^2 \times n^2$. Kot smo omenili na začetku tega razdelka, primerek igre Sudoku vsebuje n^2 podmrež, ki jih bomo označili s $S(i, j)$, kjer velja $i, j \in \{1, ..., n\}$. Nadalje definiramo še nekaj transformacij, ki jih bomo izvajali nad podmrežami:

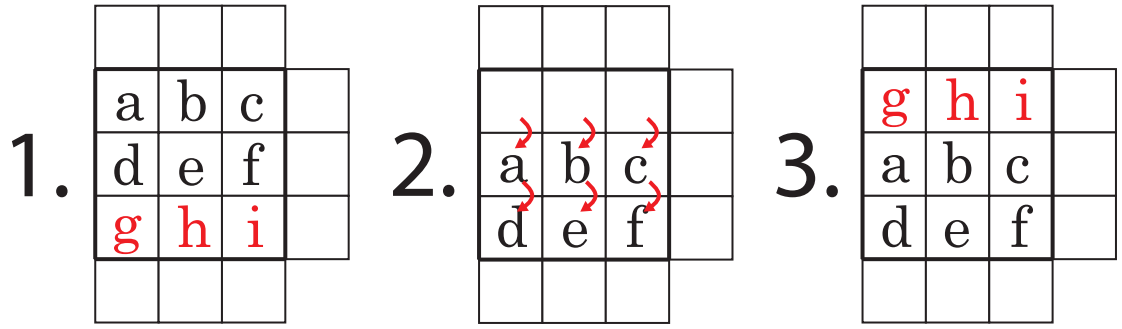
Definicija 3. *Stolpični zamik C prestavi n-ti stolpec v podmreži velikosti $n \times n$ na mesto prvega stolpca v podmreži, ostale stolpce pa zamakne v desno, tako da originalno prvi stolpec postane drugi, originalno drugi stolpec postane tretji, ... in originalno $(n-1)$-ti stolpec postane n-ti. Nadalje definiramo še C^m, ki stolpični zamik opravi m-krat, in C^0, ki podmreže ne spremeni.*



Slika 3.36: Primer stolpičnega zamika na podmreži velikosti 3×3

Primer stolpičnega zamika lahko vidimo na sliki 3.36.

Definicija 4. *Vrstični zamik R prestavi n-to vrstico v podmreži velikosti $n \times n$ na mesto prve vrstice v podmreži, ostale vrstice pa zamakne navzdol, tako da originalno prva vrstica postane druga, originalno druga vrstica postane tretja, ... in originalno $(n-1)$-ta vrstica postane n-ta. Nadalje definiramo še R^m, ki vrstični zamik opravi m-krat, in R^0, ki podmreže ne spremeni.*



Slika 3.37: Primer vrstičnega zamika na podmreži velikosti 3×3.

Primer vrstičnega zamika lahko vidimo na sliki 3.37.

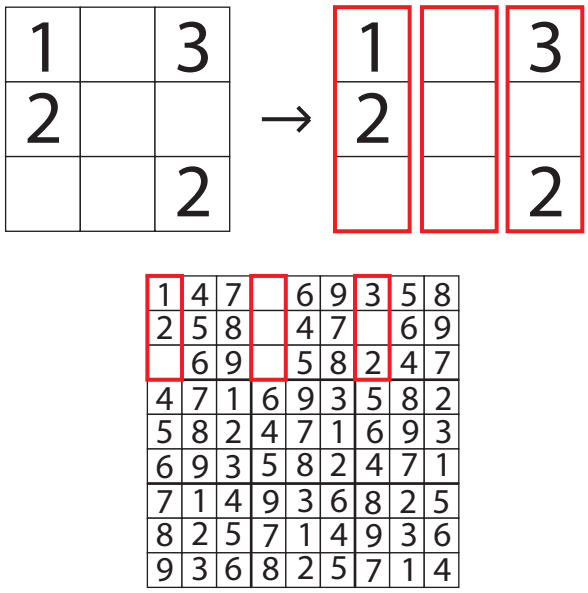
Z uporabo stolpičnega in vrstičnega zamika lahko sestavimo primerek igre Sudoku S velikosti $n^2 \times n^2$, ki pripada delnemu latinskemu kvadratu L velikosti $n \times n$ na sledeč način:

1. Podmrežo $S(1,1)$ zapolnimo tako, da velja: $S(1,1)_{i,j} = (j-1)n + i$, pri čemer je $S(1,1)_{i,j}$ vrednost celice v i-ti vrstici in j-tem stolpcu v podmreži.

2. Ostale podmreže zapolnimo tako, da velja: $S(p,q) = (R^{q-1} \circ C^{p-1})(S(1,1))$, kjer sta $p, q \in \{1, ..., n\}$.

3. Prvi stolpec podmreže $S(1,j)$ zamenjamo z j-tim stolpcem latinskega kvadrata (velja $j \in \{1, ..., n\}$).

Primer take izgradnje vidimo na sliki 3.38.



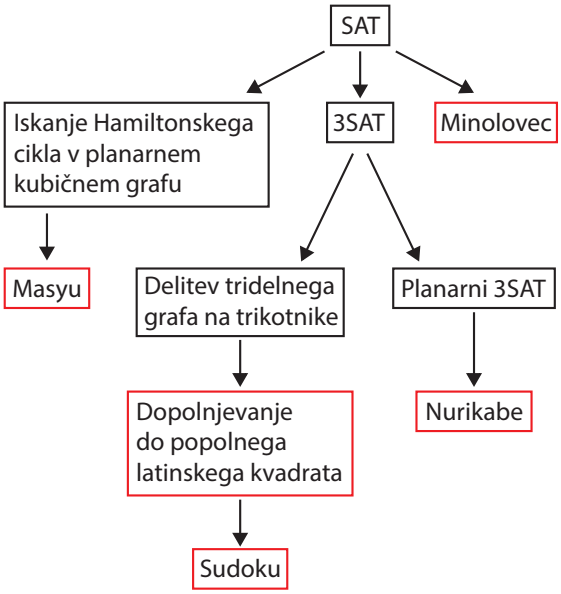
Slika 3.38: Posamezne stolpce danega delnega latinskega kvadrata velikosti 3×3 (zgoraj) vstavimo v Sudoku velikosti 9×9, ki smo ga zapolnili po danem algoritmu (spodaj).

Izgradnjo smo izvedli v polinomskem času. Iz izgradnje je očitno, da ima primerek igre Sudoku rešitev natanko tedaj, ko ima rešitev tudi delni latinski kvadrat. Vsi deli Sudoku primerka, ki niso bili zgrajeni v 3. koraku, so namreč že rešeni in nikakor ne vplivajo na reševanje praznih polj. Sudoku primerek je po nerešenih poljih povsem ekvivalenten latinskemu kvadratu.

Sedaj moramo dokazati še, da je odločitveni problem Sudoku v NP. To očitno sledi iz dejstva, da lahko za poljubno rešitev preverimo pravilnost v polinomskem času; preverimo, da se v vsakem stolpcu, vsaki vrstici in vsaki podmreži nahajajo števila od 1 do n. Iz tega sledi, da je odločitveni problem Sudoku NP-poln.

Poglavje 4

Zaključek



Slika 4.1: Slika prikazuje prevedene uganke, puščice pa predstavljajo, kako smo prišli do posameznih ugank

V diplomskem delu smo pod drobnogled vzeli zgolj pet miselnih iger (slika 4.1), dokaze NP-polnosti pa so, zlasti v zadnjih letih, dobile tudi mnoge druge. Kljub temu je področje miselnih iger in ugank izredno obsežno, prav tako pa domiselni ustvarjalci ugank stalno razvijajo nove uganke in

miselne igre. Veliko miselnih iger je po obliki in načinu reševanja podobnih NP-polnim miselnim igram, vendar nihče še ni uspel dokazati njihove NP-polnosti. Primeri takih[1] iger so:

- Kakurasu,
- Kurotto,
- Nansuke.

Kot nadaljevanje dela bi se lahko lotili raziskovanja morebitne NP-polnosti zgornjih primerov. Posebej zanimiva izgleda uganka Kurotto, ki ima veliko skupnih lastnosti z uganko Nurikabe. Te podobnosti bi lahko s pridom izkoristili pri sestavljanju prevedbe, ki je ključna za dokaz NP-polnosti.

Prav tako je zanimivo področje posplošenih miselnih iger z dvema igralcema, kot so šah, Go, Shogi in druge. Pri takih igrah se ne sprašujemo več ali ima podana igra rešitev, temveč vprašanje postane, ali je določena pozicija v igri za nas zmagovalna.

V splošnem je področje miselnih iger z vidika teoretičnega računalništva še precej neraziskano in ponuja veliko priložnosti za nova odkritja, ki bi tudi v širšem svetu lahko imela pomembne posledice.

Kot zaključno misel navajam citat madžarskega arhitekta Erna Rubika, ki je iznašel Rubikovo kocko, eno najuspešnejših ugank na svetu:

"A good puzzle, it's a fair thing. Nobody is lying. It's very clear, and the problem depends just on you.[2] *"*

[1] Vse tri omenjene igre se pojavljajo v reviji Nikoli in so dostopne na spletni strani `http://www.nikoli.co.jp/`.

[2] Dobra uganka je pravična in se nikoli ne laže. Vse je jasno in rešitev je odvisna samo od tebe.

Literatura

[1] A. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", Proceedings of the London Mathematical Society s.2 Vol. 42 p. 230–265, 1936.

[2] J. Hartmanis, "Gödel, von Neumann and the P=?NP Problem", Department of Computer Science, Cornell University, Ithaca, New York, 1989.

[3] S. Cook, "The Complexity of Theorem-Proving Procedures", University of Toronto, 1971.

[4] L. Fortnow "The Status of the P versus NP Problem", Northwestern University, Communications of the ACM, Vol. 52 No. 9, p. 78–86, 2009.

[5] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completenes", New York, 1979.

[6] M. Holzer, A. Klein, M. Kutrib, "On The NP-Completeness of The Nurikabe Pencil Puzzle and Variants Thereof", Germany, 2004.

[7] D. Lichtenstein, "Planar formulae and their uses", SIAM Journal on Computing, Vol. 11, No. 2, p. 329–343, 1982.

[8] R. Kaye, "Minesweeper is NP-complete", Mathematical Intelligencer, Vol. 22, No. 2, 2000.

[9] E. Friedman, "Pearl Puzzles are NP-complete", Stetson University, DeLand, FL 32723, 2002.

[10] M. R. Garey, D. S. Johnson, R. Endre Tarjan, "The Planar Hamiltonian Circuit Problem is NP-complete", Vol. 5, No. 4, 1976.

[11] H. De Fraysseix, J. Pach, R. Pollack, "How to Draw a Planar Graph on a Grid", Combinatorica, Vol. 10, I. 1, p. 41–51, 1990.

[12] T. Yato, T. Seta, "Complexity and Completeness of Finding Another Solution and Its Application to Puzzles", The University of Tokyo, 2003.

[13] W.D. Wallis, J.C. George, "Introduction to Combinatorics, Discrete Mathematics and its Applications, 2011.

[14] D. Singmaster, "Chronology of Recreational Mathematics", 1998.

[15] C. Colbourn, "The Complexity of Completing Partial Latin Squares", Discrete Applied Mathematics 8, 1984.

[16] R. Rizzi, "NP-complete Problem: Partition into Triangles", Corso di Complessita, Anno Accademico, 2003-2004.